Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array

Sıddıka Berna Örs[1] Lejla Batina[1,2] Bart Preneel[1] Joos Vandewalle[1]
[1]Katholieke Universiteit Leuven, ESAT/SCD-COSIC
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
{Siddika.BernaOrs, Lejla.Batina, Bart.Preneel, Joos.Vandewalle}@esat.kuleuven.ac.be
[2]SafeNet BV
Boxtelseweg 26a
5261 NE Vught, The Netherlands
lbatina@safenet-inc.com

Abstract

This paper describes a hardware architecture for modular multiplication operation which is efficient for bit-lengths suitable for both commonly used types of Public Key Cryptography (PKC) i.e. ECC and RSA Cryptosystems. The challenge of current PKC implementations is to deal with long numbers (160-2048 bits) in order to achieve system's efficiency, as well as security. RSA, still the most popular PKC, has at its root the modular exponentiation operation. Modular exponentiation consists of repeated modular multiplications, which is also the basic operation for ECC protocols. The solution proposed in this work uses a systolic array implementation and can be used for arbitrary precisions. We also present modular exponentiation based on the Montgomery's Multiplication Method (MMM).

Keywords: *Montgomery's Multiplication Method, Public Key Cryptography, RSA, ECC, FPGA, systolic array*

1 Introduction

In 1976, Diffie and Hellman introduced the idea of public key cryptography [5]. They used this concept to eliminate the need for a secure channel to exchange some secret information. Also, digital signatures were introduced which allow to uniquely bind a message to its sender. Since then, numerous public-key cryptosystems have been proposed and all these systems based their security on the difficulty of some mathematical problem. The most prominent examples are RSA, named after its inventors Rivest, Shamir and Adelman [23] and Elliptic Curve Cryptosystems (ECC), which are proposed by Koblitz [13] and Miller [18]. When comparing these two most popular public-key cryptosystems, there are several aspects to be taken into account such as: security, key lengths, speed and implementation issues. For security, the hardness of the underlying mathematical problem is essential. It is important to point out that ECC are plural equivalent security as RSA for much smaller key sizes. The reason is that all algorithms solving the mathematical problem on which ECC are based take fully exponential time. Other benefits include higher speed, lower power consumption and smaller certificates which is especially useful in constrained environments (smart cards, cellular phones, pagers etc.). The basic operation for RSA is modular exponentiation which can be realized by using repeated multiplications over $GF(p)$. The basic operation for ECC is point multiplication which also relies on efficient finite field multiplication. Commonly used finite fields in ECC protocols are $GF(p)$ and $GF(2^n)$. As a consequence, a substantial amount of research is focused on efficient and secure implementation of modular multiplication in hardware.

In 1985 Montgomery introduced a new method for modular multiplication [19]. The approach of Montgomery avoids the time consuming trial division that is a common bottleneck of other algorithms. His method is proved to be very efficient and is the basis of many implementations of modular multiplication, both in software and hardware. In this paper we look at an efficient hardware implementation of the Montgomery's modular multiplication (MMM) in FPGA.

Efficient implementation of the MMM in hardware was considered by many authors [1, 3, 4, 6, 7, 8, 9, 10, 11, 12,

14, 16, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36]. A systolic array architecture is one possibility for implementations of public key cryptography in hardware. Various solutions for systolic arrays were proposed, for example [1, 3, 4, 7, 8, 9, 10, 11, 14, 25, 28, 29, 30, 31, 35, 36].

Our contribution is in combining a systolic array architecture, which is assumed to be the best choice for hardware on current integrated circuits (ICs), with the MMM in Field Programmable Gate Array (FPGA). In this work we present the implementation results of a modular exponentiator which we designed using our MMM circuit. The implementation results for ECC using MMM can be found in [20]. Similar work was done by Blum and Paar [3]. However, their solution is less efficient because they had to use an extra step in the main algorithm for the MMM; this step was required because they do not use the optimal bound for the main parameter R (so-called Montgomery parameter) [37]. The modular exponentiation algorithm is usually repeating modular multiplication around 1500 times (assuming balanced Hamming weight of the exponent) for 1024-bit operands. Therefore, the implementation in [3] is far less efficient compared to implementation in this work.

The remainder of this paper is organized as follows. In Section 2, the underlying methods invented by Montgomery are explained in detail and we introduce common notation and parameters. We also give some comments on the bound condition for avoiding subtraction at the end of every exponentiation step introduced by Walter [37]. Section 3 gives a survey of previous work on systolic arrays and Montgomery based operations in hardware. Section 4 describes a systolic array architecture and design steps as well as the results of implementation. Conclusions and benchmarks for future work conclude the paper.

2 Previous work

This section reviews some of the most relevant previous contributions in this area. Eldridge and Walter [6] and Kornerup [14] were the first researchers who report hardware solutions for implementing the Montgomery's Multiplication Method. However, Iwamura, Matsumoto and Imai [10, 9] are the first ones to our knowledge proposing a systolic array which can execute a modular exponentiation operation using Montgomery modular multiplication.

Tenca and Koç introduced a pipelined Montgomery multiplier, which has the ability to work on any given operand precision and is adjustable to any chip area in [26]. Savaş *et al.* used the same design methodology to obtain a dual-field multiplier, i.e., the unit which can handle multiplication in both types of finite fields in [24]. That multiplier would have obvious benefits for many applications of public key cryptography.

Iwamura, Matsumoto and Imai [11] considered the usual bottleneck for hardware implementations of Montgomery's algorithm, i.e., the fact that the number of output bits may exceed the number of input bits. They derived the bound $R \geq 2^{n+2}$ for $R = 2^r$. Hence, they concluded that $r = n + 2$ is the minimum possible value for which the examination of the size of the output each time the Montgomery method is executed, may be omitted. This property is desired in order to be able to feed back directly the result of each multiplication. Here, *n* is the maximal number of bits of N, $N < 2^n$. This bound can be further improved to the condition $R > 4N$, which is according to Walter [37], the best possible bound in practice. The work of Walter offers many useful results for Montgomery's techniques. In [34], which is further improved in [37], he showed that the Montgomery exponentiation method requires no final subtraction, which is very important for fast implementations. Detailed review of previous work can be found in [2].

Relevant previous work considering FPGA is presented by Blum and Paar [3]. The latency of processing elements used to construct the systolic array introduced in that work is higher than the processing elements introduced in this work. This difference brings to our work higher clock frequency, hence results in a fast implementation. We also improve on the work of [3] by giving a practical implementation of the most recent theoretical work on the bound. More precisely, in their work the Montgomery parameter R is set as $R = 2^{n+3}$. We use $4N < R = 2^{n+2}$. In this way, the number of repetitions for Montgomery's algorithm is only $n + 2$ for radix 2 implementations, compared with $n + 3$. In the case of higher radix it can perform multiplication in $\lceil \frac{n+2}{\alpha} \rceil$ as explained in [1]. Also this architecture is equally suitable for both types of cryptography, ECC as well as RSA. Note that, the same authors have reported an implementation with high-radix in [4].

3 Montgomery Modular Multiplication

For modular multiplication Montgomery's technique is chosen [19]. Montgomery multiplication is defined as follows:

$$Mont(x, y) = xyR^{-1} \mod N \tag{1}$$

For a word base $b = 2^\alpha$, R should be chosen such that $R = 2^r = (2^\alpha)^l > N$. There is a one-to-one correspondence between each element $x \in \mathbb{Z}_\mathbb{N}$ and its Montgomery representation $xR \bmod N$. This Montgomery representation allows very efficient modular arithmetic especially for multiplication. Montgomery's method for multiplying two integers x and y (called N-residues) modulo N, avoids division by N which is the most expensive operation in hardware. The method requires conversion of x and y to an N-residue domain and conversion of the calculation result back to $\mathbb{Z}_\mathbb{N}$. The procedure is as follows. To compute $Z = xy \bmod N$,

one first has to compute the Montgomery multiplication of x and $R^2 \bmod N$ to get $Z' = xR \bmod N$. $Mont(Z', y)$ gives the desired result. When computing the Montgomery product $T = Mont(x, y) = xyR^{-1} \bmod N$, the procedure shown in Algorithm 1 is performed [17].

In the original notation of Montgomery after each multiplication a reduction was needed (Step 7 in Algorithm 1). The input had the restriction $X, Y < N$ and the output T was bounded by $T < 2N$. As a consequence, if $T > N$, N must be subtracted so that the output can be used as input of the next multiplication. To avoid this subtraction a bound for R is known [37] such that for inputs $X, Y < 2N$ the output is also bounded by $T < 2N$.

In [34] the need of avoiding reduction after each multiplication is addressed. In practice this means that the output of the multiplication can be directly used as an input of the next Montgomery multiplication. We want to find a bound on R such that with $X, Y < 2N$ the output of the Montgomery multiplication $T < 2N$. Write $R \geq kN$, then:

$$T = \frac{XY + mN}{R} = \frac{XY}{R} + \frac{m}{R}N < \frac{4}{k}N + N \quad (2)$$

where, $m = (XY \mod R)N' \mod R$ [1].

Hence, $T < 2N$ for $k \geq 4$, implying: $4N \leq R$. We will use $4N < R = 2^{l+2}$, by taking $\alpha = 1$ for simplicity and making the iteration starting from Step 2 execute $l + 2$ times. As the result of the decision for α, $-N^{-1} \bmod 2^\alpha$ can be written as $(2 - n_0)^{-1} \bmod 2$. Because N is odd for RSA and an odd prime for ECC, $n_0 = 1$ which results to $N' = 1$. We will use the Algorithm 2 for MMM which includes these improvements.

Algorithm 1 Montgomery modular multiplication with final subtraction

Require: $N = (n_{l-1} \cdots n_1 n_0)_{2^\alpha}$, $x = (x_{l-1} \cdots x_1 x_0)_{2^\alpha}$, $y = (y_{l-1} \cdots y_1 y_0)_{2^\alpha}$ with $x, y \in [0, N-1]$, $R = (2^\alpha)^l$, $gcd(N, 2^\alpha) = 1$ and $N' = -N^{-1} \mod 2^\alpha$

Ensure: $xyR^{-1} \mod N$

1: T $\leftarrow 0$.
2: **for** i from 0 to $(l-1)$ **do**
3: $\quad m_i \leftarrow (t_0 + x_i y_0)\, N' \bmod 2^\alpha$
4: $\quad T \leftarrow (T + x_i y + m_i N)/2^\alpha$
5: **end for**
6: **if** $T \geq N$ **then**
7: $\quad T \leftarrow T - N$
8: **end if**
9: Return (T)

All the operations will be done modulo $2N$ through the modular exponentiation. The final round in the modular exponentiation is the conversion to the integer domain, i.e., calculating the Montgomery multiplication of the last result and 1. The same arguments as above prove that this final step remains within the following bound: $Mont(T, 1) \leq N$. In practice, $A^B \mod N = N$ will never occur since $A \neq 0 \bmod N$ [1].

Algorithm 2 Montgomery modular multiplication without final subtraction

Require: $N = (n_{l-1} \cdots n_1 n_0)_2$, $x = (x_l \cdots x_1 x_0)_2$, $y = (y_l \cdots y_1 y_0)_2$ with $x, y \in [0, 2N-1]$, $R = 2^{l+2}$, $gcd(N, 2) = 1$

Ensure: $xyR^{-1} \mod 2N$

1: $T \leftarrow 0$
2: **for** i from 0 to $l+1$ **do**
3: $\quad m_i \leftarrow (t_0 + x_i y_0) \bmod 2$
4: $\quad T \leftarrow (T + x_i y + m_i N)/2$
5: **end for**
6: Return (T)

4 Hardware Implementation

4.1 Design Overview

Our system can be divided hierarchically into three levels:

1. Systolic Array Cell: computes 1 bit of T in Step 4 of Algorithm 2.
2. Systolic Array: computes one iteration of Step 2 of Algorithm 2.
3. Montgomery Modular Multiplication Circuit (MMMC): computes complete Algorithm 2.
4. Modular Exponentiator: combines modular multiplications to realize modular exponentiation according to Algorithm 3.

In the following sections we have described the system using a bottom-up approach.

4.2 Systolic Array Cells

The i-th iteration of Step 2 in Algorithm 2 computes the temporary results

$$T_i = 2^{-\alpha}(T_{i-1} + x_i \times Y + m_i \times N + 2) \quad (3)$$

where $i = 0, \cdots, l+1$ and $T_{-1} = 0$ [35]. The j-th digit of T_i is obtained using the recurrence relation

$$2^2 \times c1_{i,j} + 2 \times c0_{i,j} + t_{i,j} = t_{i-1,j+1} + x_i \times y_j + m_i \times n_j + 2 \times c1_{i,j-1} + c0_{i,j-1} \quad (4)$$

$i = 0, \cdots, l+1$, $j = 0, \cdots, l$, $c1_{i,-1} = 0$ and $c0_{i,-1} = 0$. In Eq.(4), $2 \times c1_{i,j} + c0_{i,j}$, $j = -1, \cdots, l$, denotes the carry chain up the adder.

The regular cell of the systolic array consists of two full-adders (FA), one half-adder (HA) and two AND-gates as shown in Fig. 1.(a). We can calculate m_i by the following equation:

$$m_i = (t_{i-1,1} + x_i \times y_0) \bmod 2 = t_{i-1,1} \oplus x_i \times y_0 \quad (5)$$

$i = 0, \cdots, l+1$ and $t_{-1,1} = 0$. Here m_i is not an input to the rightmost cell, but obtained in the rightmost cell.

Because there is no carry input to the rightmost cell, the equation for calculating $t_{i,0}$ can be simplified as shown by Eq. (6).

$$2 \times c0_{i,0} + t_{i,0} = t_{i-1,1} + x_i \times y_0 + m_i \quad (6)$$

$i = 0, \cdots, l+1$ and $t_{-1,1} = 0$. By combining Eq. (5) and Eq. (6), it can easily be shown that $t_{i,0} = 0$ and the equation for calculating $c0_{i,0}$ is as follows:

$$c0_0 = t_{i-1,1} + x_i \times y_0 \quad (7)$$

$i = 0, \cdots, l+1$ and $t_{-1,1} = 0$. The rightmost cell of the systolic array consists of one AND, one OR and one XOR gate as shown in the Fig. 1.(b).

Because there is only one carry input from rightmost cell, Eq.(4) can be simplified for $t_{i,1}$ as follows, which is obtained by the cell shown in Fig. 1.(c). It consists of one FA, two HAs and two AND-gates.

$$2^2 \times c1_{i,1} + 2 \times c0_{i,1} + t_{i,1} = t_{i-1,2} + x_i \times y_1 + m_i \times n_1 + c0_0 \quad (8)$$

$i = 0, \cdots, l+1$ and $t_{-1,2} = 0$.

Because $n_l = 0$, the equation of $t_{i,l}$ can be simplified as follows:

$$\begin{aligned} 2 \times t_{i,l+1} + t_{i,l} = \quad & t_{i-1,l+1} + x_i \times y_l \\ & + 2 \times c1_{i,l-1} + c0_{i,l-1} \end{aligned} \quad (9)$$

$i = 0, \cdots, l+1$ and $t_{-1,l+1} = 0$. This equation is implemented by the l-th cell, which is shown in Fig. 1.(d). This cell consists of one FA, one AND and one XOR-gate.

The i-th row computes T_i from T_{i-1}. Each cell operates in a single clock cycle. Then the i, j-th cell processes the digits of Eq.(4) at clock cycle time $2i + j$.

4.3 Systolic Array

To obtain a linear, pipelined modular multiplier, only one row of cells is taken. The j-th cell behaves like cell (i, j), computing Eq.(4) at time $2i + j$ for $i = 0, \cdots, l+1$.

The schematic view of the systolic array is shown in Fig. 2. $X(0)$ denotes the least significant bit (LSB) of the register in which the input X is stored. T denotes the intermediate value register. The carry chain is stored in the $C0$ and $C1$ registers.

Fig. 2 shows that the T_{j+1} output of $(j+1)$-th cell is used as an input for j-th cell during the next iteration. This way the division by 2 in Step 4 of Algorithm 1 is realized.

Total area of the systolic array is $(5l-3)XOR + (7l-7)AND + (4l-5)OR$ gates and $4l$ flip-flops. The critical path is the same as the critical path of one regular cell and it is independent of the bit length of the operands. So it is $2T_{FA}(c_{in} \rightarrow c_{out}) + T_{HA}(c_{in} \rightarrow c_{out})$.

4.4 Modular Montgomery Multiplication Circuit

The MMMC has three l-bit data inputs X, Y and N, one START instruction input, one DONE output, which indicates that the operation is ended, and an l-bit RESULT output.

The MMMC is designed using the algorithmic state machine (ASM) approach. For detailed information about ASM approach, reader is referred to [15]. The circuit consists of a controller and a data path as shown in Fig. 3. The controller has four states, IDLE, MUL1, MUL2 and OUT. The data path consists of a systolic array, four internal registers, a counter and a comparator. The controller stays in

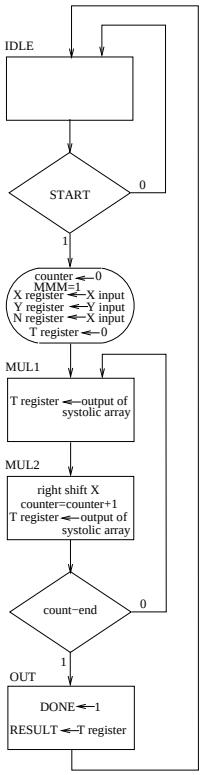

Figure 4. Algorithmic state machine of Montgomery modular multiplier

the IDLE state waiting for the START instruction. When

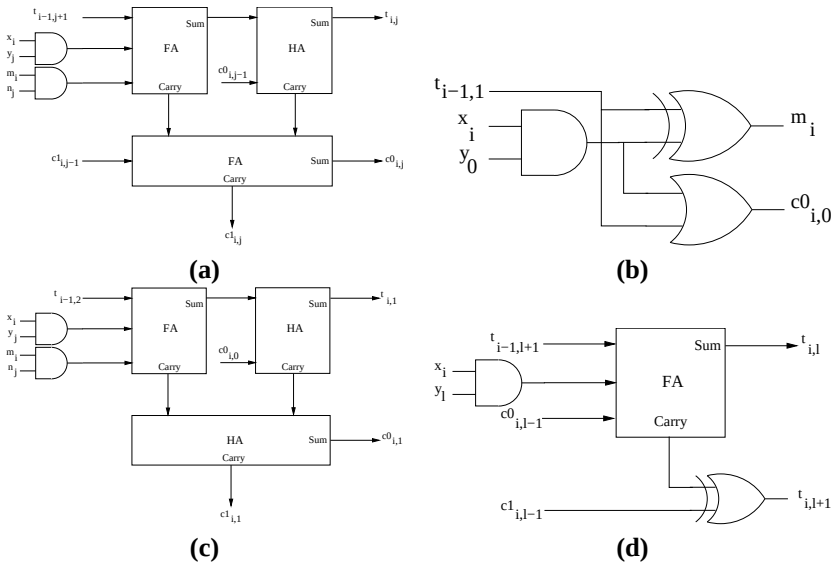

Figure 1. Schematic view of systolic array cells; (a) Regular, (b) Rightmost, (c) 1-st bit, (d) Leftmost

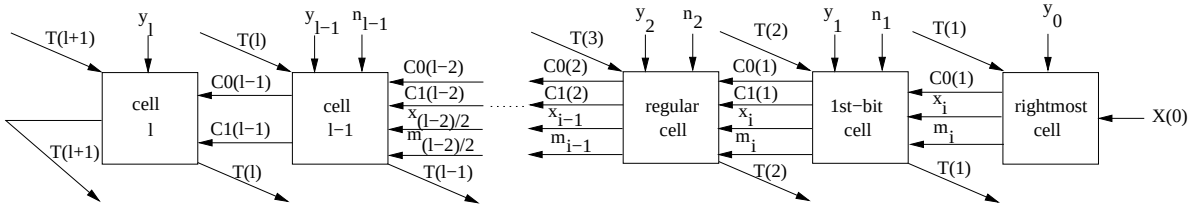

Figure 2. Schematic view of complete systolic array

the START input is set, X, Y and N registers are loaded by input values, the T register and the counter are reset.

In MUL1, the outputs of the systolic array cells are written to the T register and controller goes to the MUL2 state. When the controller is in the MUL2 state, the counter is incremented by 1. When the counter value reaches $2(l+1)$, the comparator sets the "count-end" signal. Then the controller goes to the OUT state in which the value of the T register is written to the RESULT output and the acknowledgement signal DONE is set.

In the MUL2 state, the X register is shifted one bit right and the most significant bit (MSB) of the X register is filled 0. This ensures that, during the last iteration of Step 2 of Algorithm 2, the value of $X(0)$ will be 0.

As mentioned before $t_{i,j}$ is calculated at the $(2i+j)$-th clock cycle, $i = 1, \cdots, l+2$ and $j = 1, \cdots, l$. $t_{l+2,l}$ is calculated at the $(2(l+2)+l)$-th clock cycle. Hence, the total number of clock cycles for completing one modular Montgomery multiplication equals $3l+4$.

In the previous work from Blum and Paar [4] the cells process u-bit data in one clock cycle. The 3-bit control registers are put in the cells to control the output of four complex multiplexors. These bring high latency on the critical path of the one cell and as a consequence, the clock frequency is lower. In this work, cells process 1-bit in one clock cycle, the circuit is constructed only using combinational elements and the architecture is much more simpler as shown in Fig. 1.

In [4], total number of bits used for control logic is $3\lceil l/u \rceil$-bit. The role of the control in their circuit and how they implemented the complete algorithm using it is not clear. In this work, the control logic is $\log_2(l+2)+2$-bit, including 2-bit state register, $\log_2(l+2)$-bit counter and a comparator. It is implemented separately from systolic array as shown in Figure 3 and controls the execution of the modular Montgomery multiplication algorithm according to the ASM shown in Figure 4.

Implementation Results of The Modular Montgomery Multiplication Circuit: The MMMC is implemented on Xilinx V812E-BG-560-8 (Virtex E) FPGA. Number of slices (S), clock period (T_p), time-area product (TA) and time for one MMM (T_{MMM}) for different bit

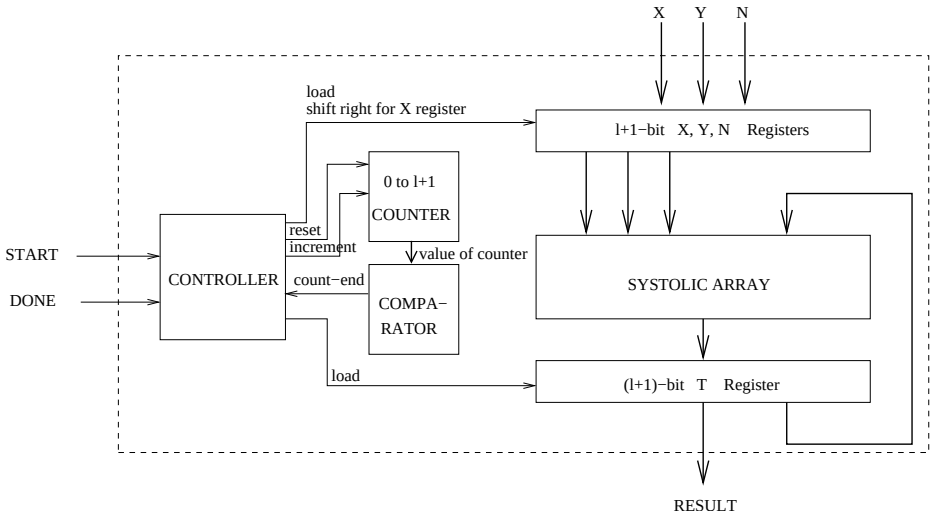

Figure 3. Architecture of the Montgomery modular multiplier circuit

length l are given in Table 2.

As it is shown in Table 2, the clock frequency is independent from the bit length. This property gives our circuit the advantage of suitability to various applications with different bit lengths like RSA and ECC.

4.5 Modular Exponentiation and RSA

The RSA algorithm is based on modular exponentiation. The private key of a user consists of two large primes p and q and an exponent D. The public key consists of the modulus $N = p \cdot q$ and an exponent E such that $E = D^{-1} \bmod lcm((p-1),(q-1))$. To encrypt a message M the user computes: $C = M^E \bmod N$. Decryption is done by $M = C^D \bmod N$. Modular exponentiation can be realized by using the standard square and multiply algorithm as given by Algorithm 3 [17].

Algorithm 3 Modular exponentiation

Require: Integers N, $0 \leq M < N$, $0 < E < N$, $E = (e_{t-1}, e_{t-2}, \cdots, e_0)_2$, $e_{t-1} = 1$
Ensure: $M^E \bmod N$
1: $A \rightarrow M$
2: **for** i from $t-2$ to 0 **do**
3: $A \rightarrow AA \bmod N$
4: **if** $e_i = 1$ **then**
5: $A \rightarrow AM \bmod N$
6: **end if**
7: **end for**
8: Return (A)

If MMMC is used for multiplication then the result will be with an extra factor $R^{-1} = 2^{-(l+2)}$. This is compensated by pre-Montgomery multiplying M by $R^2 \bmod N$, so that $MR \bmod N$ is fed into the exponentiator. The square in Step 3 of Algorithm 3 will be $Mont(AR, AR) = A^2R \bmod N$ and the multiplication in Step 5 of Algorithm 3 will be $Mont(AR, MR) = AMR \bmod N$. Hence the result of the modular exponentiation will be $M^E R \bmod N$. The only post-processing is then another Montgomery multiplication by 1, which removes R.

The pre-computation is done in $2(2(l+2)+1)+l = 5l+10$ clock cycles. One multiplication or square takes $3l+4$ clock cycles. If all the bits of E are 1 then the complete exponentiation takes $2l(3l+4)$ clock cycles. If only one bit of E is 1 then it takes $l(3l+4)$. The post-processing takes $2+l$ clock cycles. So the complete timing of modular exponentiation, $T_{mod-exp}$ can be given by the following inequality:

$$3l^2 + 10l + 12 \leq T_{mod-exp} \leq 6l^2 + 14l + 12 \qquad (10)$$

l is the number of bits in N. The average times needed for one modular exponentiation for different bit lengths are given in Table 1.

5 Conclusions and Future Work

We have described an efficient systolic array architecture for modular multiplication which is basic operation for public key cryptosystems as RSA and ECC. We use the method of Montgomery, which is proven to be very efficient and secure in hardware. Namely, the optimal bound is achieved which, with some savings in hardware, omits completely all reduction steps that are presumed to be vulnerable to side-channel attacks. Also we realize a modular exponentiator

Table 1. Clock period (T_p) and average time for one modular exponentiation ($T_{mod-exp}$) for different bit length l on Xilinx V812E-BG-560-8

l	T_p ns	The average $T_{mod-exp}$ ms
32	9.256	0.046
128	10.242	0.775
256	9.956	2.974
512	10.501	12.468
1024	10.458	49.508

Table 2. Number of slices (S), clock period (T_p), time-area product (TA) and time for one MMM (T_{MMM}) for different bit length l on Xilinx V812E-BG-560-8

l	# S	T_p ns	TA $S \cdot ns$	T_{MMM} μs
32	225	9.256	2082.6	0.926
64	418	9.221	3854.38	1.807
128	806	10.242	8255.05	3.974
256	1548	9.956	15411.88	7.686
512	2972	10.501	31208.97	16.171
1024	5706	10.458	59673.35	32.168

which uses repeating modular multiplications. We implemented our architecture on Xilinx V812E-BG-560-8 (Virtex E) FPGA which is very useful to try efficiently different design choices, i.e., different algorithms for modular multiplication and exponentiation for less expense compared to ASIC.

One direction in which this work should go is to implement also an ECC basic operation, i.e., point multiplication. This operation does not require modular exponentiation but modular multiplication only, so all required components are available. A cryptographic device dealing with both types of PKC would be very useful to secure communication systems.

Acknowledgements

Sıddıka Berna Örs is funded by a research grant of the Katholieke Universiteit Leuven, Belgium. Dr. Bart Preneel and Dr. Joos Vandewalle are professors at the Katholieke Universiteit Leuven, Belgium. This work was supported by Concerted Research Action GOA-MEFISTO-666 of the Flemish Government and by the FWO "Identification and Cryptography" project (G.0141.03).

References

[1] L. Batina and G. Muurling. Montgomery in practice: How to do it more efficiently in hardware. In B. Preneel, editor, *Proceedings of RSA 2002 Cryptographers' Track*, number 2271 in Lecture Notes in Computer Science, pages 40–52, San Jose, USA, February 18-22 2002. Springer-Verlag.

[2] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle. Hardware architectures for public key cryptography. *Elsevier Science Integration the VLSI Journal*, in print, 2002.

[3] T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. In *Proceedings of 14th IEEE Symposium on Computer Arithmetic*, pages 70–77, Adelaide, Australia, April 14-16 1999.

[4] T. Blum and C. Paar. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers*, 50(7):759–764, July 2001.

[5] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[6] S. E. Eldridge and C. D. Walter. Hardware implementation of Montgomery's modular multiplication algorithm. *IEEE Transactions on Computers*, 42:693–699, 93.

[7] S. Even. Systolic modular multiplication. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology: Proceedings of CRYPTO'90*, number 537 in Lecture Notes in Computer Science, pages 619–624. Springer-Verlag, 1990.

[8] W. L. Freking and K. K. Parhi. Performance-scalable array architectures for modular multiplication. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 149–160. IEEE, 2000.

[9] K. Iwamura, T. Matsumoto, and H. Imai. High-speed implementation methods for RSA scheme. In R. A. Rueppel, editor, *Advances in Cryptology: Proceedings of EUROCRYPT'92*, number 658 in Lecture Notes in Computer Science, pages 221–238. Springer-Verlag, 1992.

[10] K. Iwamura, T. Matsumoto, and H. Imai. Systolic-arrays for modular exponentiation using Montgomery method. In R. A. Rueppel, editor, *Advances in Cryptology: Proceedings of EUROCRYPT'92*, number 658 in Lecture Notes in Computer Science, pages 477–481. Springer-Verlag, 1992.

[11] K. Iwamura, T. Matsumoto, and H. Imai. Montgomery modular multiplication method and systolic arrays suitable for modular exponentiation. *Electronics and Communications in Japan*, 77(3):40–50, 1994.

[12] Y. S. Kim, W. S. Kang, and J. R. Choi. Implementation of 1024-bit modular processor for RSA cryptosystem. In *Proceedings of Asia-Pasific Conference on ASIC (AP-ASIC)*, Cheju Island, Korea, August 28-30 2000.

[13] N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[14] P. Kornerup. A systolic, linear-array multiplier for a class of right-shift algorithms. *IEEE Transactions on Computers*, 43(8):892–898, August 1994.

[15] M. M. Mano and C. R. Kime. *Logic and Computer Design Fundamentals*. Prentice Hall, Upper Saddle River, New Jersey 07458, second edition, 2001.

[16] W. P. Marnane. Optimised bit serial modular multiplier for implementation on field programmable gate arrays. *Electronics Letters*, 34(8):738–739, April 1998.

[17] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[18] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO'85*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.

[19] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, Vol. 44:519–521, 1985.

[20] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle. Hardware implementation of an elliptic curve processor over GF(p). 2002. Submitted for publication.

[21] H. Orup. Simplifying quotient determination in high-radix modular multiplication. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 193–199. IEEE, 1995.

[22] J. Poldre, K. Tammemae, and M. Mandre. Modular exponent realization on FPGAs. In *Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm (FPL'98)*, pages 336–347, Tallinn, Estonia, August 31-September 3 1998.

[23] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[24] E. Savaş, A. F. Tenca, and Ç. K. Koç. A scalable and unified multiplier architecture for finite fields GF(*p*) and GF(2^m). In C. Paar and Ç. K. Koç, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2000)*, number 1965 in Lecture Notes in Computer Science, pages 281–296. Springer-Verlag, 2000.

[25] C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu. An improved Montgomery's algorithm for high-speed RSA public-key cryptosystem. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 7(2):280–284, June 1999.

[26] A. F. Tenca and Ç. K. Koç. A scalable architecture for Montgomery multiplication. In Ç. K. Koç and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999)*, number 1717 in Lecture Notes in Computer Science, pages 94–108. Springer-Verlag, 1999.

[27] A. F. Tenca, G. Todorov, and Ç. K. Koç. High-radix design of a scalable modular multiplier. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2001)*, number 2162 in Lecture Notes in Computer Science, pages 189–205. Springer-Verlag, 2001.

[28] A. A. Tiountchik. Systolic modular exponentiation via Montgomery algorithm. *Electronics Letters*, 34(9):874–875, April 1998.

[29] E. Trichina and A. Tiountchik. Scalable algorithm for Montgomery multiplication and its implementation on the coarse-grain reconfigurable chip. In D. Naccache, editor, *Proceedings of Topics in Cryptology - CT-RSA 2001*, number 2020 in Lecture Notes in Computer Science, pages 235–249. Springer-Verlag, 2001.

[30] W.-C. Tsai, C. B. Shung, and S.-J. Wang. Two systolic architectures for modular multiplication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):103–107, February 2000.

[31] C. D. Walter. Systolic modular multiplication. *IEEE Transactions on Computers*, 42:376–378, 1993.

[32] C. D. Walter. Still faster modular multiplication. *Electronics Letters*, 31(4):263–264, February 1995.

[33] C. D. Walter. Techniques for the hardware implementation of modular multiplication. In *Proceedings of 2nd IMACS International Conference on Circuits, Systems & Computers*, volume 2, pages 945–949, Athens, October 1998.

[34] C. D. Walter. Montgomery exponentiation needs no final subtraction. *Electronic letters*, 35(21):1831–1832, October 1999.

[35] C. D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In Ç. K. Koç and C. Paar, editors, *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 1999)*, number 1717 in Lecture Notes in Computer Science, pages 80–93. Springer-Verlag, 1999.

[36] C. D. Walter. An improved linear systolic array for fast modular exponentiation. *IEE Computers and Digital Techniques*, 147(5):323–328, September 2000.

[37] C. D. Walter. Precise bounds for Montgomery modular multiplication and some potentially insecure RSA moduli. In B. Preneel, editor, *Proceedings of Topics in Cryptology- CT-RSA 2002*, number 2271 in Lecture Notes in Computer Science, pages 30–39, 2002.